Genetic Algorithms for the Assembly Line Balancing Problem: A Real-World Automotive Application

Solivan Arantes Valente[1], Heitor Silvério Lopes[2], Lúcia Valéria R. de Arruda[2]

[1]UnicenP - Centro Universitário Positivo, Curitiba PR, Brazil
[2]CEFET-PR - Centro Federal de Educação Tecnológica do Paraná, Curitiba PR, Brazil

Abstract. This paper reports the use of Genetic Algorithms (GAs) to solve the assembly line balancing problem in a real-world application: a car assembly facility. The problem is modeled and a standard GA is applied. The line layout solution found by GA reduces by 28.5% the total assembly time of the current line layout, which implies in a significant reduction of costs. This result suggests that the use of GAs in real-world industrial problems can be very promising.

1 Introduction

A Genetic Algorithm (GA) is a well-known search and optimization method based on Darwin's principle of evolution of the species and on some basic fundamentals of genetics [3]. From the classifying point of view, GA belongs to the area of Evolutionary Computation, which also includes Genetic Programming, Classifier Systems, Evolution Strategies and Evolutionary Programming.

The main scope of GAs is optimization, and many engineering problems are optimization problems. Therefore, along the last decades GAs have been successfully applied to several problems in several Engineering fields. In optimization, the goal is to find a particular set of values for the variables of the problem that minimizes a cost function or maximizes a gain function. There are numerous optimization methods, numerical and algebraic, that search for a set of optimal values in the multidimensional search space of the problem at hand. However, in problems where non-linearity, noise, discontinuity or extremely large search spaces are present, those methods may be computationally unfeasible or may not apply at all. It is in this scenery of multimodal problems that are hardly handled by classic methods that GAs come as a simple and efficient alternative.

R. Roy et al. (eds.), *Soft Computing and Industry*

2 Genetic algorithms and the assembly line balancing problem

A strong argument for the use of GAs in the well-known industrial assembly line balancing problem is the huge search space of this class of combinatorial problem. Even the balancing of a simple line may become a quite hard and slow task for a deterministic algorithm, given the size of the search space, as we shall exemplify below.

In this work, the last stage of a larger automotive assembly facility is optimized. All data used in this work are real and the name of company is purposely omitted. The part of the assembly line considered here is composed by 10 workstations. At each one of them workers can perform their jobs at both sides of the line, left and right, which totalizes 13 activities to be done.

A brief analysis of the line topology (see Fig. 1) shows that the number of possible layout configurations is given by a permutation $P_{2n,p}$, i.e., (2n!)/(2n-p)!, where "n" is the number of workstations and "p" is the number of activities to be performed. The assembly line under analysis has n=10 and p=13, which results in about $4.83.10^{14}$ possible arrangements for the line layout. If one expands the planning horizon to include the whole final line of this same facility, in which there are 66 activities to be done in 49 workstations, the number of possible configurations raises to about $3.58.10^{118}$. This search space is untreatable by any sequential search method, in acceptable time, regardless the computational power available.

The optimization of the working time in an industrial facility has some obvious consequences. The productivity growth reflects directly in the company's profit, allowing recovery of the initial investments in a shorter time. Besides, it increases the company's ability to respond to the market demands. In this paper it is reported how the assembly line balancing problem can be modeled and successfully solved with a standard GA.

Some work have already been developed using GAs in optimization of combinatorial problems in industry, including variations of the classical job shop/flow shop problems, as well as more complex planning problems. See, for instance, Chan & Hu [2], Knosala & Wal [5], Kopfer [6], Li & Love [7], Li et al [8], Nakano [9], Reeves [10], and Warwick and Tsang [11]. The solution of the assembly line balancing problem with GAs has been particularly treated by Anderson and Ferris [1]. In this work the authors propose the variables coding and the precedence matrix that, with minor changes, were used in the current work. However, they implemented a parallel GA, which was not the case here.

3 The Assembly Line Optimization Problem

The assembly line under analysis is depicted in Fig. 1. From left to right, vehicles being assembled enter the line and pass through ten workstations (P_1 to P_{10}). At each one of them there may be activities (A_i) at one or both sides of the line. The line must not be completely compacted, i.e., it must not have activities at both

sides of the line at all workstations due to space restrictions caused by the machinery and logistics shelves placement.

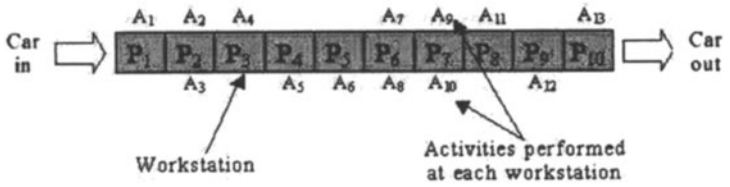

Fig. 1. Representation of the production line under analysis

Each workstation has a fixed length (about 5m) and the vehicle moves on softly, without stopping along the process. The total working time at a given workstation is represented by its longest activity, since both sides start at the same time. For example, the total working time of, say, P_2 workstation is given by the longest duration of activities A_2 and A_3.

The total working time of this line section is given by the sum of all workstations' working times. In the real line, not yet optimized, the current time is 21min 22s (or 21.37min).

4 Solution using genetic algorithms

Variables encoding

Figure 2 shows the chromosome encoding adopted in this work. The chromosome is composed by 13 integer numbers, one for each activity to be performed in the line, all between 1 and 10. The content of each "gene" in the chromosome identifies one of the 10 workstations, and the position it occupies maps the activities performed at that workstation. For instance, in Fig. 1, activities A_2 and A_3 are both performed at workstation P_2, activity A_6 at workstation P_5, and so on.

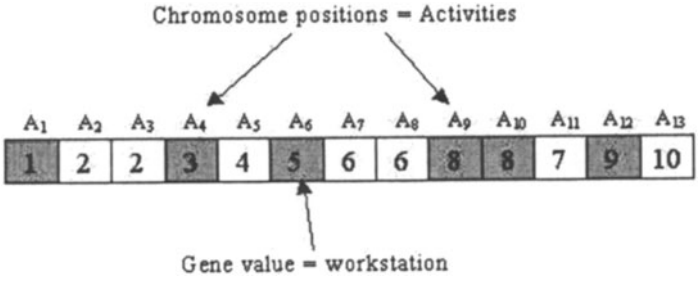

Fig. 2. Chromosome encoding

Although this is a combinatorial problem, the special crossover operators proposed by Goldberg [3], such as PMX, OX and CX, do not apply since the precedence order between the activities does not follow directly their numbering.

dence order between the activities does not follow directly their numbering. For example, activity A_{11} must be performed before activity A_9 (see Table 2) even though it appears in a rightmost position of the chromosome. Thus, the information of the relative position of a gene with respect to another is not relevant. This is a special feature of such approach comparing to other implementations found in the literature.

Relevant data

Two fundamental information are needed to model the problem: the time necessary to perform every activity and the precedence order between them. The data shown in Tables 1 and 2, from the real production line, represent, respectively, the execution timing for each activity and the precedence among activities.

In Table 1, Effective Working Time – EWT is the time necessary for the operation to be performed by the worker. Variable Operations Time – VOT is the time necessary for other associated operations (e.g., the time needed for the worker to get a part in the shelf); and the Total Activity Time – TAT is the sum EWT+VOT. Notice that the time fractions are given in hundreds of minutes, so as to facilitate arithmetic operations, since conversions are avoided.

Table 2 shows the precedence relations between activities. For a given pair of activities, A_i and A_j, there is precedence between them if there is a "1" in the intersection of the line pointed by A_i and the column pointed by of A_j. Otherwise there is no precedence.

For instance, consider activities A_9 and A_{11} (as mentioned in 'Variables encoding' section). Table 2 shows that there is precedence between these activities (in this order), meaning that activity A_{11} must be performed before activity A_9. The reciprocal is somewhat obvious, which means that a position filled with a "1" must have its symmetric counterpart filled with a "0".

Table 1. Execution time of activities

Activity	Time (minutes)		
	Effective Working Time (EWT)	**Variable Operations Time (VOT)**	**Total Activity Time (TAT)**
A_1	0.91	0.24	1.15
A_2	1.84	0.53	2.37
A_3	1.64	0.43	2.07
A_4	0.99	0.31	1.30
A_5	2.14	0.56	2.70
A_6	1.89	0.63	2.52
A_7	1.78	0.28	2.06
A_8	1.46	0.56	2.02
A_9	1.72	0.52	2.24
A_{10}	1.58	0.37	1.95

A_{11}	1.81	0.46	2.27
A_{12}	1.70	0.52	2.22
A_{13}	2.17	0.37	2.54

Table 2. Precedence between pairs of activities

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}
A_1	0	0	0	0	0	0	0	0	0	0	0	0	0
A_2	1	0	0	0	0	0	0	0	0	0	0	0	0
A_3	1	0	0	0	0	0	0	0	0	0	0	0	0
A_4	1	1	1	0	0	0	0	0	0	0	0	0	0
A_5	1	0	0	0	0	0	0	0	0	0	0	0	0
A_6	1	0	0	0	1	0	0	0	0	0	0	0	0
A_7	1	0	1	0	0	0	0	0	0	0	0	0	0
A_8	1	0	0	0	1	0	0	0	0	0	0	0	0
A_9	1	0	1	0	1	0	1	1	0	0	1	0	0
A_{10}	1	0	1	0	1	0	1	1	0	0	1	0	0
A_{11}	0	0	0	0	0	0	0	0	0	0	0	0	0
A_{12}	1	0	1	0	1	0	1	1	0	0	0	0	0
A_{13}	1	0	1	0	1	0	1	1	0	0	0	1	0

The use of a table to represent the precedence orders between activities has some advantages: first, it allows a quick overview by inspection, second, it allows a faster correction of any precedence that should ever be altered due to process modifications, and finally it provides faster calculations of the violations when the GA has to compute the penalty function.

Objective and fitness functions

As mentioned before, the total working time of a workstation is given by its longest activity since both start at the same time. If there's only one activity associated to the workstation, its working time is given by this activity duration. Workstations without activities are given null (0) work time.

The objective function is computed as the sum of the working times of all workstations, given by equation (1), where $TAT[A_{left}(P_i)]$ is the Total Activity Time of the activity assigned to the left side of the i^{th} workstation and $TAT[A_{right}(P_i)]$ is its equivalent to the right side.

$$F_{obj} = \sum_{i=1}^{10} \max\left\{TAT[A_{left}(P_i)], TAT[A_{right}(P_i)]\right\} \quad (1)$$

Finally, the fitness function (to be maximized) is computed by equation (2), taking into account the penalties (to be explained in the next section). Its value is then normalized, i.e. forced to be in the interval [0,1]:

$$F_{fitness} = 1 - \frac{F_{obj} + C_p.Penalty}{STT + 72.Tp_{\max}} \quad (2)$$

Where *Cp* is a penalty application coefficient (set to 2.0), *72* is the maximum possible number of violations[1] and *STT* is the sum of the duration time for all activities, given by equation (3).

$$STT = \sum_{i=1}^{13} TAT(A_i) \quad (3)$$

Restrictions and Penalties

During the evolutionary process not all individuals represent feasible solutions. Nevertheless, as pointed out by Anderson and Ferris [1] it is interesting to keep some of these "unfit" individuals in the population in order to maintain the genetic diversity. Thus, the so called "candidate solutions" that don't represent adequate solutions to the optimization problem are better "punished" instead of simply eliminated. The restrictions imposed to the candidate solutions are:

a) There must be no more than two activities at each workstation. This restriction is due to the physical limitations of the production line, since workers can be at its left and right sides. (Remark: there are workstations where activities are performed under or over the vehicle; however, in order to simplify the planning, activities are always assigned to the side of the line where the workers stand).

b) The order of the activities must obey the precedence shown in Table 2.

When such restrictions are not fully satisfied, penalties are imposed to the candidate solution. This reflects in a fitness decrement for that individual as a function of the violation to the restrictions above, according to the criteria:

a) Any individual having more than 2 activities at a workstation is given maximal penalty: its fitness is set to 0. Since this restriction is a physical limitation, it is not interesting to have candidate solutions like this.

b) Individuals that respect the first restriction and violate the precedence order among activities are punished proportionally to the number of violations, according to equation (4):

$$Penalty = V.Tp_{max} \quad (4)$$

---

[1] From the expression 11*2+9*2+7*2+5*2+3*2+1*2=72, which is obtained from the worst case, where the 2 activities assigned to the first workstation violate all the precedence relative to the other 11 activities, the 2 activities assigned to the second workstation violate all the precedence relative to the 9 activities left, and so on.

Where V is the number of precedence violations and Tp_{max} is the duration of the longest activity in the line.

GA parameters

After several experiments with different combinations of parameters for the GA, the best results were achieved with the parameters described in the sequence. The two classical genetic operators were used: one-point crossover (with probability of 80%) and simple bit mutation (with probability of 4% per bit). Elitism was used throughout generations so as to always keep the best individual. Since elitism may cause a harder selective pressure, a more efficient selection method than the classical roulette wheel was used: stochastic universal sampling. The population size was set to 100 individuals with generation gap 1, meaning that a whole new population was created every generation. The criterion used to stop the evolutionary process was simply timing-out a given number of generations, set to 200.

The development of the algorithm was based in a former version of the software package named *Galopps* (Genetic Algorithm Optimized for Portability and Parallelism), developed by Goodman [4], and written in ANSI C.

5 Results

The evolution of fitness is shown in Fig. 3. The "x" axis represents the generation number and the "y" axis represents the fitness value (between 0 and 1). The upper trace is the fitness of the best individual and the lower trace is the average fitness of the whole population. Every run took around 0.85s of processing time in a Pentium ™ III 500MHz under Windows ™ 98.

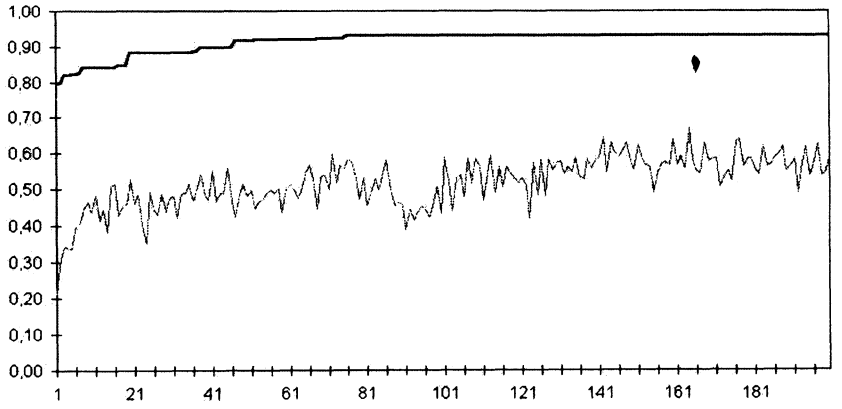

Fig. 3. Fitness evolution

For the evaluation of the best individual found in each run of the GA, a Microsoft Excel ™ worksheet was built in order to compute the total working time associ-

ated with the line layout represented by such individual. This worksheet also analyses the chromosome mapping and produces a sketch of the line layout. Part of the worksheet is shown in Fig. 4.

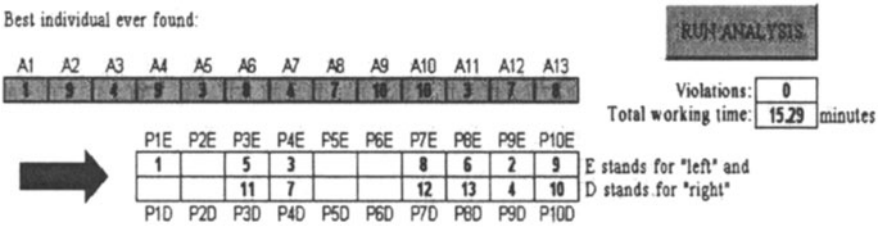

Fig. 4. Worksheet for analysis of the best individual

After running the GA several times, the best individual ever found was the one represented in Fig. 5. This individual represents a solution for the assembly line layout, which is sketched in Fig. 6. The best solution found leads to a total working time of 15.29min, which represents a 28.5% reduction of the current line configuration time (21.37min).

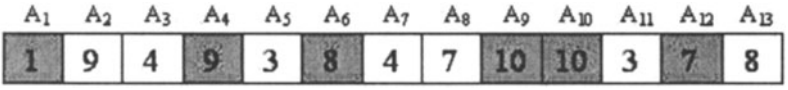

Fig. 5. Best individual ever found

Fig. 6. Best solution ever found (line layout)

6 Conclusions

We have shown with this work that GAs can be very useful to find better alternatives for the layout of an assembly line. If such analysis had been used during the facility design phase, it could have suggested improvements in the layout of the whole production line, optimizing the overall working time.

Considering the dimensionality of the search space and the combinatorial nature of the problem, GAs were found to be very efficient and fast. A comparison with another classical methodologies for the line-balancing problem shall be done in the future to investigate how competitive a GA is.

For the sake of simplicity, the present work did not consider logistics issues, such as parts shelves disposal. It was not analyzed, as well, the costs due to the layout changes in the current line. Notwithstanding, the solution found is real and shows

that GAs can be used as a powerful tool for the industrial organization, with a significant gain in time that can reflect in the overall productivity and, possibly, in profit increases for the company.

References

1. Anderson, E.J. & Ferris, M.C. (1994). Genetic algorithms for combinatorial optimization: the assembly line balancing problem. *ORSA Journal of Computing*, vol. 6, no. 2, pp. 161-173.
2. Chan, W.T. & Hu, H. (2000). Precast production scheduling with genetic algorithms. *Proc. of 2000 Congress on Evolutionary Computation*, vol. 2, pp. 1087-1094.
3. Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.
4. Goodman, E.D. (1996). An Introduction to Galopps – The Genetic ALgorithm Optimized for Portability and Parallelism System. *Technical Report #96-07-01*, Michigan State University, East Lansing.
5. Knosala, R. & Wal, T. (2001). A production scheduling problem using genetic algorithms. *Journal of Materials Processing Technology*, vol. 109, no. 1-2, pp. 90-95.
6. Kopfer, H. (1996). *Evolutionary search and the job shop: investigations on genetic algorithms for production scheduling*. Heidelberg: Physica-Verlag.
7. Li, H. & Love, P. (1998) Site-level facilities layout using genetic algorithms. *Journal of Computing in Civil Engineering*, vol. 12, no. 4, pp. 227-231.
8. Li, Y. & Ip, W.H. & Wang, D.W. (1998) Genetic algorithm approach to earliness and tardiness production scheduling and planning problem. *International Journal of Production Economics*, vol. 54, no. 1, pp. 65-76.
9. Nakano, R. (1991). Conventional genetic algorithm for job shop problems. *Proc. of the 4th International Conference on Genetic Algorithms*, pp. 474-479.
10. Reeves, C.R. (1995). A genetic algorithm for flowshop sequencing. *Computers in Operations Research*, vol. 22, no. 1, pp. 5-13.
11. Warwick, T. & Tsang, E.P.K. (1996). Tackling car-sequencing problems using a generic genetic algorithm. *Evolutionary Computation*, vol. 3, no 3, pp. 267-298.